多变体执行误报问题分析与解决方法综述 *

席睿成[1,2]，张　铮[1,2†]，朱鹏喆[1,2]，刘子敬[1,2]

(1. 中国人民解放军战略支援部队信息工程大学，网络与空间安全学院，郑州 450001;2. 网络通信与安全紫金山实验室，南京 211100)

摘　要：从安全角度出发，多变体执行(Multi-variant Execution，MVX)被广泛应用于网络安全防御，但多变体执行存在一个共性问题：即各路执行体向裁决器返回内容时，合路产生的误报难以解决。排除机器环境等客观因素，产生误报是因为表决器收到合路信息后开始对非一致变量做安全判断，除真实攻击造成的非一致变量外，还夹杂着正常系统运行产生的非一致变量(如内存描述符、端口号、随机数、代码及进程内的线程调用顺序)，从而造成表决器误判，影响多变体系统正常运行。如果能降低多变体执行的误报率，则可以有效地提高系统效率及防御能力。对近年来多变体执行的类型进行归类，并对多变体执行产生的误报问题及解决策略进行归纳总结，分析多变体执行产生裁决误报的原因，选择 Pina 算法进行同步的策略、编译器模块插桩的策略、缩小表决边界的策略，对三种方案在特定应用场景下实验分析，分析每个方法的功能及性能，指出各自策略的优点及缺点。最后讨论现有多变体执行现有技术未解决的难点和未来的研究方向。

关键词：多变体执行；安全判断；非一致变量；系统误报

中图分类号：TP309.2　　doi: 10.19734/j.issn.1001-3695.2022.02.0093

Summary of analysis and solution of multivariate technology false alarm

Xi Ruicheng[1,2], Zhang Zheng[1,2†], Zhu Pengzhe[1,2], Liu Zijing[1,2]

(1. *Information Engineering University*, *Cyber & Space Security School College*, *Zhengzhou* 450001, *China*; 2.*Purple Mountain Laboratories*, *Nanjing* 211100, *China*)

Abstract: From a security view, Multi-variant Execution (MVX) is widely used in network security defense, but there is a common problem in multi-variant execution: When each executive body returns content to the arbiter, it is difficult to solve the false alarm caused by the combination. Excluding objective factors such as the machine environment, false alarms are generated because the voter starts to make security judgments on non-consistent variables after receiving the combined information. In addition to non-consistent variables caused by real attacks, there are also non-consistent variables generated by normal system operation. (such as memory descriptors, port numbers, random numbers, codes, and the calling sequence of threads in the process) , voted normal operation of the multi-variant system made misjudgment. If reduced the false alarm rate of multi-variant execution, the system efficiency and defense capability would improved effectively. This paper classifies the types of multi-variant execution in recent years, summarizes the false alarms caused by multi-variant execution and the solutions, Analyze the causes of false alarms caused by multi-variant execution, select the Pina algorithm for synchronization, the compiler module instrumentation strategy, and the strategy for narrowing the voting boundary, analyzed the three schemes in specific application scenarios, and analyzed each method. Then pointed the functions and performance of each strategy, pointed the advantages and disadvantages of each strategy. Finally, discussed the unsolved difficulties and future research directions of the existing multi-variant implementation

Key words: multi-variant execution; security judgments; inconsistent variables; systematic false alarm

1　研究背景及意义

1.1　软件开发中存在安全隐患

在系统级开发过程中，由于编程语言的局限性与软件开发者缺乏安全知识，应用程序会存在例如整型溢出、悬空指针、缓冲区溢出等漏洞，在开发过程中，由于大量的使用第三方库、借鉴开源的项目源代码，虽然提高了效率，但也带来了安全隐患问题。如软件漏洞中最常见的进程控制劫持流攻击，是攻击者利用内存信息泄露绕过保护机制，获得重要信息地址，向程序注入 payload，窃取敏感数据。多变体执行是有效抵御进程控制劫持流攻击的方法之一。

1.2　多变体执行的介绍

多变体执行的架构由执行体和监控模块两部分组成，多变体可以表现为多个进程或者多个线程。多变体执行技术[1,2]，既可以保护系统免受内存破坏攻击，也可以提高系统的运行效能，其关键思想是同步运行多个不同的执行体，为它们提供相同的输入，并监控它们的运行以发现分歧。

从安全防御的角度[3~6]考虑，如果正在运行的执行体之间的内部差异，导致对恶意输入的响应出现明显的差异，监视器可以在执行中检测这种差异，然后发出警报或终止执行，从而达到防御的效果。该技术通过强制或监视变体系统调用的锁步执行，透明地执行相同输入，接收各执行体执行后的

收稿日期：2022-02-22；修回日期：2022-04-24　　基金项目：国家自然科学基金资助项目(61521003)

作者简介：席睿成(1997-)，男，宁夏石嘴山人，硕士研究生，主要研究方向为多变体执行技术、主动防御等；张铮(1975-)，男（通信作者），湖北黄冈人，副教授，网络通信与安全紫金山实验室双聘教授，硕导，主要研究方向为高性能计算、拟态防御等(370722031@qq.com)；朱鹏喆(2000-)，男，河南驻马店人，硕士研究生，主要研究方向为主动防御、编译器技术等；刘子敬(1998-)，男，山东临朐人，硕士研究生，主要研究方向为内生安全、多变体执行技术等.

结果，监视器可以检测到执行中的差异，然后发出警报。多变体执行框架如图 1 所示。

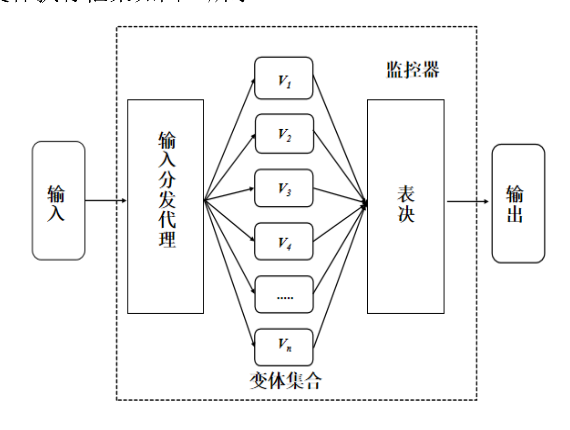


图 1　多变体执行框架

Fig. 1　Multi-variant execution framework

1.3　动态异构冗余概念

随着多变体执行的发展，动态冗余异构[7,8](dynamic heterogeneous redundancy，DHR)思想应运而生，该思想在多变体执行的基础上增加动态和反馈的特征，文献[9,10]对动态冗余异构进行了论述。DHR 主要由五部分组成，分别是输入模块、处理模块、输出模块、构建模块、调度模块[11~13]。输入模块负责将用户输入分成多份，分发给各执行体；处理模块负责将各执行体执行后的多份输出发送至表决器；输出模块负责表决收到的多份输出，判定若存在 n 个或 n 个以上的结果一致，则输出结果，否则，截断输出过程。构建模块负责构建执行体集；调度模块负责动态选择算法，从执行体集中选择执行体进行调度，在执行体调度完成后，将服务体下线，清洗回滚至原始状态。动态冗余异构框架如图 2 所示。

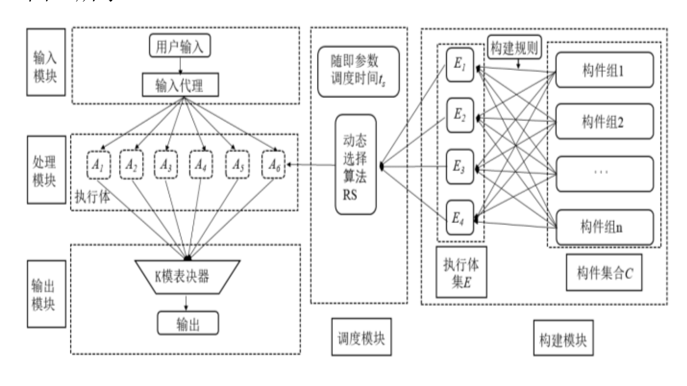


图 2　动态冗余异构框架

Fig. 2　Dynamic redundancy heterogeneous framework

每个执行体结构不一样但功能等价，表决器通过对比各执行体之间的信息差异来判断是否被攻击。一旦表决器判定当前受到攻击，则调用动态选择算法，从备份的异构组件中挑选合适的执行体对异常的执行体进行替换，通过表决并进行负反馈至调度模块。DHR 具有随机性、动态性、多样性，从外界看，这些内生的特性使信息系统能够以黑盒[14~16]的角度存在，应对不确定的攻击，这极大的提高了攻击者的难度，理论上，当多变体系统各执行体不存在共同漏洞时，该系统是足够安全的。

1.4　多变体执行的发展

在多变体提出之前，基于操作系统的多样化思想，设计了多样化机器和系统调用层级的映射随机化。还存着基于软件的多样化代码，比如程序级别的指令集随机化[17~20]。随着 N-Variant 的出现，越来越多国内外团队开始研究多变体执行，经过十几年的发展，多变体执行蓬勃发展。本文选取的相关文献以多变体执行架构的研究机构和团队为主线，下面简要介绍十几种主要类型的多变体执行系统发展：

在 2006 年，Cox 等人[21]首次针对软件安全问题提出了 N-Variant 执行架构，它定义理想 MVX 系统安全特性[22~25] 的等效项和检测机制。该架构使用内存空间的异构和指令集标签化技术来生成执行体。每个执行体都相当于独立的进程，缺点是，该系统无法支持多线程冗余程序执行，且因功能不完善，限制系统调用的使用。同年，Berger 提出 DieHard 多变体执行架构[26]，在该架构里，通过改造每个执行体的堆对象，使其异构。每个执行体接收输入后,将结果输出到一个共享的内存区域,该区域中每个执行体有自己的专属缓冲区,表决进程在提前规划好的同步点上对共享内存区域的输出进行比较,进而输出正确的结果。缺点是，该系统不能防御基于堆栈损坏的攻击行为[27,28]　。

2007 年，Cavallaro[29]多变体架构基于地址空间分区(Address Space Partitioning，ASP)的原理将框架中一个执行体的地址空间“移位”k 字节，使得两个执行体的地址空间的相对距离发生改变，从而实现多样化的目的，进而阻止部分地址覆盖攻击。缺点是，该系统的局限性在于不能抵抗类如相对寻址这类攻击。

2009 年，Salamat 等人提出 Orchestra[30]、多变体架构，基于堆栈结构的特点，设计出两个堆栈反向生长的执行体，该框架中的监视模块作为单独进程运行，使用 ptrace API 来对执行体的系统调用进行拦截和比对,不需要对操作系统进行任何改动。缺点是，拦截比对系统调用消耗性能巨大，需要频繁的对上下文进行切换。

2003 年，哥特大学 Stijn Volckaert 团队设计出 GHUMVEE[31]监视器，GHUMVEE 和其他面向安全的多变体框架的监控组件都是依赖于 ptrace API 来监视执行体[32,33]，在用户空间，作为一个单独的进程运行，监控需要同步的系统调用，能正确处理多线程复制和异步信号传递。缺点是频繁切换上下文。

2015 年，Hosek 团队设计出 Varan[34]多变体执行框架，首次引入异步共享机制，类似于内存中的记录重放框架，其中一个执行体充当领导者，直接执行系统调用，将结果写入共享的环形缓冲区。其他版本执行体为追随者，只是从环形缓冲区中读取结果。监控器放在每个执行体内部,能够极大的减少跨进程的上下文切换，加载内核模块拦截系统调用以检测执行体的行为，并统一执行体的输出。监视器充当中间组件，向内核报告执行体的状态，并在执行体崩溃或检测到执行体行为差异时执行恢复异常处理。Varan 将选择性二进制重写[35]与高性能事件流相结合，提供一种灵活高效的用户空间解决方案，但没有解决执行体间系统调用产生的误报问题。

2016 年 Stijn Volckaert 团队引入监控模块 IP-Mon[36,37]。Remon 结合跨进程监控器和进程内监控器的优点，Remon 其中的进程内监视器在程序内输入，而跨进程监视器强制执行对有潜在危险的系统调用的锁步执行；另一方面，安全的系统调用在没有外部监控的情况下进行，从而提高系统效率。该系统将监控模块放入应用程序进程里， 使用“记录+重放”方法提供对多线程的支持。记录+重放方法，也称为领导者/追随者方法，该方法通过初始化，设立主执行体从执行体，监控器捕获主执行体中执行同步操作的顺序，并在其他执行体中强制执行相同的顺序。

2016 年，Koning 等人[38]从虚拟化技术，基于硬件虚拟化的系统模式 Dune，提出 MvArmor 的多变体架构，改善多变体架构的性能瓶颈，并通过自定义的安全策略[39]来把握性能和安全之间的平衡，该架构能够通过环境感知来动态

生成执行体。遗憾的是 Dune 这一虚拟化框架并不是线程安全的，而且并未得到持续更新。

2018 年 LUK 等人基于安全防御的角度提出 BUDDY[40]的多变体执行架构，在 I/O 操作设置同步点，监控输出口，当攻击者向外发送数据时对端口进行监控，很大效率的提升架构安全防御能力，减少监控同步比较的次数。监控器拦截系统调用后，将结果从同步缓冲区返回给各执行体，尽可能使整个系统损耗性能达到最小，能抵御信息泄漏，该系统缺陷是，不能有效地解决执行体之间的误报问题。

2019 年，SalamatB 等人基于操作系统内核[41]，提出了把监控器放在内核里的 Kmvx 架构，在内核空间里生成两个执行体，具体实现方式是在同一机器上运行两个多样化编译后的内核执行体，并构造两个完全不相交的虚拟内存映射，在执行系统调用时，两个内核同时处理，对执行结果进行同步检查以判断是否存在内核内存泄漏。该架构可以防止内核中的信息泄露问题，不过仍然具有较大的性能开销，部分功能还待探究。

2019 年同年，Alexios Voulimeneas 等人设计 DMON 框架[42]，DMON 将一组执行体分布在一组异构的物理机器上，执行体异构体现包括不同的指令集、字符顺序、调用规则、系统调用接口，以及潜在的硬件安全特性差异。DMON 设计两种监视器:L-MON 监视器监控主执行体，而从执行体由自带的 F-MON 监视器监督。只要变体执行系统调用，这些组件就会交互，每当主执行体或从执行体试图进入或退出系统调用时，相应的 L-MON 或 F-MON 中断并挂起变体的状态，读取被中断的系统调用的调用号，并调用监控进程中的专用处理程序例程，该例程的作用是为各变体的系统调用实现检查逻辑和复制逻辑。监视器在进入系统调用时中断，检查处理程序。在 F-MON 中，收集关于变体状态的信息，将该信息发送给 L-MON，并等待 L-MON 确认从执行体处于等同于主执行体的状态。在 L-MON 中，检查处理程序等待来自 F-MON 的传入状态信息，将该状态信息与主执行体的状态进行比较，并将比较结果通知 F-MON。

2020 年，Xiaoguang Wang 等人设计 MonGuard 系统[43]，MonGuard 是一个保护进程内监视器和库的系统。MonGuard 利用 Intel MPK 高效更新内存访问权限。MonGuard 实现只执行内存和代码随机化来隐藏监控代码，使用它来实现一个受保护的进程内 MVX 监视器。其实验结果表明，MonGuard 通过合理的组件内隔离可以大大提高监控性能。

2020 年，潘传幸等人在拟态防御原理[1,2,44,45]的基础之上实现系统 MimicBox[46]； 当发现冗余执行的进程有数据流出虚拟内存空间时，表决器主动进行表决。MimicBox 使用 ptrace 系统调用实现系统调用的拦截、替换、表决内容提取、返回值覆盖等功能，通过第三届拟态强网杯精英挑战赛的拟态 pwn 题验证拟态执行的防御有效性，但存在表决器误报的情况，主要原因是 MimicBox 中冗余的执行体之间存在如进程号、随机数、文件描述符等参数不一致的情况，影响表决器裁决。

多变体执行总结如表 1 所示。

2 多变体执行误报的产生原因

多变体执行系统产生误报的本质原因：在多执行体冗余执行过程中，本身会提供一些存在不确定性的参数，此类参数的在执行过程中，其属性不可预测。如随机数，时间戳等。由于被多变体执行改造的系统拥有异构冗余性，因此当用户发起请求时，多个变体可能产生响应不一致的情况，从而造成监控模块表决误判。

多变体框架的误报问题不可避免，是多变体系统自身冗余异构后带来的潜在问题，误报问题根据粒度不同，表现方式则不同，下面举几个常见的例子来阐述误报问题及其危害：

表 1 多变体执行总结

Tab. 1 Multivariant executive summary

体执行架构	变体生成技术	监控器实现	是否考虑多线程程序场景	是否解决随机数误报
N-variant[21~25]	地址空间分区，指令集标签化	内核自定义模块	否	否
DieHard[26~28]	堆布局随机化	内核自定义模块	否	否
Cavallaro[29]	非重叠地址空间	ptrace	否	否
Orchestra[30]	改变栈的生长方向	ptrace	是	否
GHUMVEE[31~33]	完全支持 ASLR	ptrace	否	否
VARAN[34,35]	多个程序修正	二进制重写	是	是
ReMon[36,37]	完全支持 ASLR，不相交的代码布局	内核自定义模块 + ptrace	是	否
MvArmor[38,39]	不重叠的地址空间和堆分配偏移	Dune	否	否
BUDDY[40]	分区地址随机化，堆栈随机填充	内核	否	否
Kmvx[41]	构造两个完全不相交的虚拟内存映射	内核自定义模块	否	是
Dmon[42]	分布式系统，使用网络进行通信	ptrace	否	否
MonGuard[43]	地址空间随机化	ptrace	是	否
MimicBox[44~46]	地址空间随机化	ptrace	是	否

2.1 情况一 文件读写引起的误报

文件读写案例如图 1 所示。

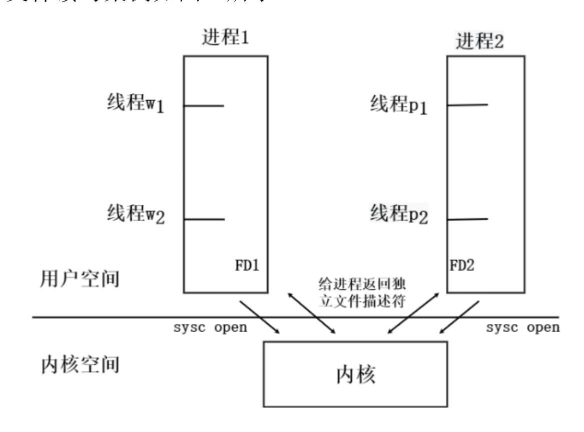


图 3 文件读写案例图

Fig. 3 File reading and writing case diagram

Linux 操作系统的本质是用文件去定义一切，因此文件描述符的作用在系统的作用至关重要，多数的系统调用都需要利用文件描述符去执行。由于进程描述符是进程独有的资源，在不同进程之间不共享。

假设系统里，有两个进程，把它们看成两个执行体，每个进程里又有两个线程，现在系统里要进行打开文件操作，由于进程是拥有资源的最小单位，线程是调度的最小单位，线程是进程中的一个执行实体，实际操作是由线程完成的，即打开文件进行 sysc open 的原语指令是由进程中的某一个线程去操作，但具体由哪个线程去操作，执行顺序是不确定的。当进程进行打开文件操作时，如果没有指定某个线程进行打开文件，进程的两个线程会进行争抢系统调用 sysc open 操作，在多变体执行的框架里，为保证表决时的正确性，需要维持主从执行体的输入一致。在普通的操作下，打开单个文件，不需要考虑多线程的问题，但如果要打开两个文件时，需要进程内的两个线程进行争抢，很容易发生混乱。如果两个执行体之间的线程调度不同，那么它们传到表决器的行为也会有差异， 进程的 PID 在整个系统中是唯一的，但线程的 PID 不同 。

该误报属于表决器裁决引起的误报，是由于在拟态化改造改造的过程中对不需要进行异构的部分进行异构处理，导致多个执行体正常执行的结果也存在不一致，最终被表决器错误地裁决为攻击，严重影响多变体执行系统的可用性。多变体执行系统依赖于通过软件多样化技术生成的多个功能等价但存在异构性的多个进程执行体，虽然通过地址空间配置随机加载 ASLR[47]等随机化技术处理后，同一个程序多次运行时功能能够保持一致，但在操作系统层面仍可能仍存在不一致的信息，这种不一致主要体现在程序打开的文件描述符对应的文件信息和进程 ID 等方面。

2.2 情况二 多线程产生的误报

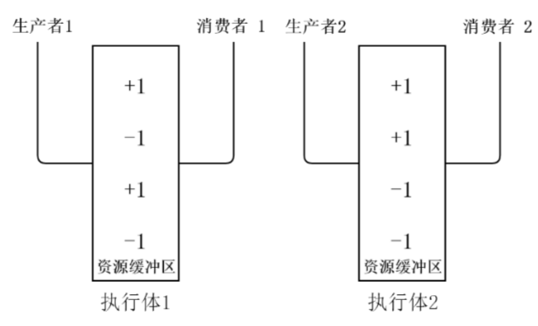


图 4 生产者消费者案例图

Fig. 4 Producer consumer case diagram

情况二，为日常的生产者消费者案例（图 2），如：假设多线程冗余执行的环境里，主执行体和从执行体分别都有一个缓冲区，只有在生产者生产资源将资源放到缓冲区后，消费者才能去消费资源，当缓冲区没有资源时，消费者不能消费。生产者生产一次资源，记录为+1，消费者消费一次资源，记录为-1。在执行程序时，主执行体缓冲区资源发生+1，-1，+1，-1。而从执行体缓冲区资源发生+1，+1，-1，-1 的操作，在显示结果的时候，二者达到一样的功能，但是在实现过程中，进行不同的修改,在监控组件的表决器中，如果监控粒度到达系统调用级别时，则可能在表决器表决时会产生不一致的结果。

2.3 情况三 随机数误报

以 Openssl 中的随机模块为例，Openssl[48,49]是一个加密解密的安全通信协议模块，能够生成随机性很强的口令，发送数据的两端通过一对密钥来进行加密解密，对会话进行安全保护[50]。如使用公钥来加密用户发给服务端的会话密钥，然后使用私钥来解密会话密钥。在 SSL 交互过程中，需要生成随机数，Openssl 会通过对系统内部数据计算摘要来生成随机数，在多变体执行框架下，每个执行体的 Openssl 模块都会产生独立的随机数，而在表决器裁决的时候，如果表决粒度过大，会发生误判，把执行体分发过来的随机数当成不一致的参数，从而会导致误报。随机数误报如图 5 所示。

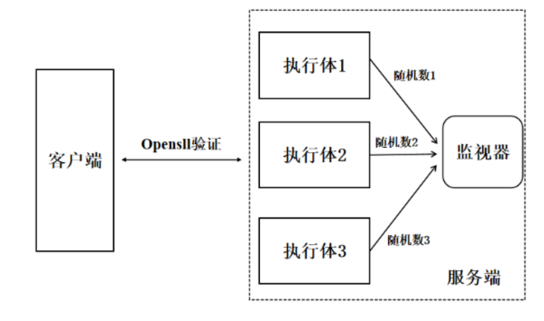


图 5 随机数误报例图

Fig. 5 Random number false positive example

2.4 情况四 代码异构的误报（图 6）

在代码函数里，为实现某一特定功能，不同的编程人员会因为不同的编写习惯及不同的函数特性，写出不同内容，但结果输出一样的代码[51]。

strace hello 1	strace hello 2
write(1,"hello world",11)	write(1,"hello ",6) write(1,"world",5)

图 6 代码异构案例图

Fig. 6 Code heterogeneous case diagram

在早前的多变体执行架构中，大多采用粗粒度，比较最后各执行体输出结果进行表决判断，这一现象并没有得到人们的重视。但随着工程的发展，引入多线程等概念，多变体执行架构的监视器监控点变得更多，监控粒度变得更细，表决不只是比对各执行体结果，还需要比对更细的粒度，如各执形体各自的系统调用。若某个多变体执行架构采用更细的粒度，监控执行体的系统调用层，在系统调用层级就开始做同步表决，则会出现误报问题，如图中的两块代码，左边代码输出 hello world 只需要一行，调用一次 sys_write()操作，右边代码输出 hello world 需要调用两次 sys_write()操作，在监控器上，表决的判断规则如果是比对系统调用的信息，很容易出现误报。

3 多变体执行误报的解决策略

3.1 Pina 同步算法

Pina 等人基于软件多样性提出一种规则语言 Domain-Specific Languagr(DSL) [52]用以解决执行体之间因系统调用号执行顺序不同导致产生良性误报的问题。在初始化多变体执行环境时，该架构使用一个监视进程来拦截所有执行体发出的系统调用。两个不同版本的执行体规则匹配如图 7 所示。

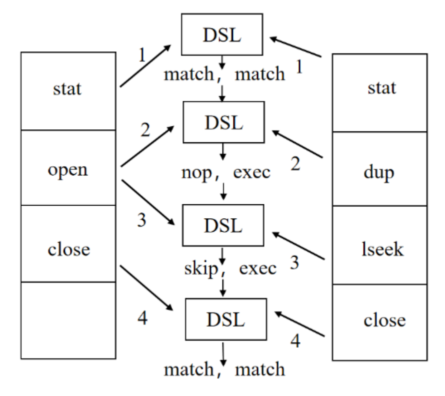


图 7 两个不同版本的执行体规则匹配

Fig. 7 Two different versions of the body rule match

DSL 解决执行体之间因系统调用执行顺序的不同产生

的误报问题，从系统调用跟踪日志开始手动编写所需的规则，这些日志分别从每个版本获得

具体比较方法：令其中一个版本担任“领导者”，称为记录端，直接执行系统调用，并将它的结果写入共享环缓冲区。其他版本称之为追随端，追随端从环缓冲区读取记录端的系统调用顺序，从而完成自己的系统调用。主从执行体两端在每一次执行系统调用时，算法进行 match 比对，看看两端的语句是否等效，第一句的 stat 两边等效，匹配成功，进入下一步，左边是 open，右边是 dup+lseek，此时并没有 match 成功，这时候需要用事先写好的规则来进行判断，open 是否等效于 dup+lseek，当判断出两者等效，则开始 skip 操作，跳入下一步系统调用比较，下一步发现左右两端都是 close，成功匹配成功。

在章节 2.4 案例中，提到的代码异构的案例，就是典型的 open 操作和 dup+lseek 操作，Pina 等人还提出一种从系统调用跟踪[53]对中自动提取 DSL 规则的算法。，DSL 语言判断文中的 LSR 为记录端，RSR 为追随端，DSL 表述在两个系统调用序列之间操作记录和重放的规则。在每个步骤中，对于每一个序列，DSL 将下一个系统调用作为输入，将要采取的操作作为输出。对于记录序列和重放序列之间的每次系统调用(图 7 中的步骤 1 和步骤 4)，DSL 通过 MATCH 匹配双方，从而将两个序列向前推进一个位置。

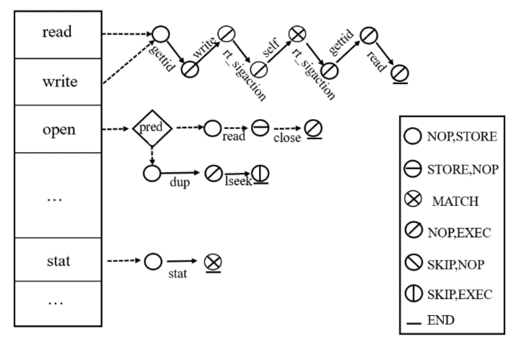


图 8　Pina 同步算法匹配流程图

Fig. 8　Pina synchronization algorithm matching flow chart

图 8 为匹配过程中的顺序流程，空圆圈表示 Nop，Nop 是空操作指令，用于控制时间周期，Nop 是 no operation 的缩写，执行 Nop 指令只使程序计数器 PC 加 1，所以占用一个机器周期。EXEC 系统调用并没有创建新的进程，只是替换原来进程上下文的内容，代码段，数据段，堆栈段被新的进程所替代。

3.2　编译器模块插桩的策略

微观到线程概念里，即便使几个执行体同样的输入，但如果执行体之间进行线程调度的顺序不同，那么它们外部可见的行为也可能会不同，导致表决时产生不必要的差异。这类良性误报问题对多变体多线程执行产生影响，毕竟多变体执行环境本身是依靠检测差异来检测攻击行为。

下面介绍一种针对多变体执行的进程中的多线程乱序执行导致误报的解决方案

Stijn Volckaert 团队采取的是，设计编译器插桩方法“Taming Parallelism in a Multi-Variant Execution Environment”[54]，最容易实现的是在系统调用级别上执行同步，在 sync 操作前后，插桩自己设计的同步库，以保证线程交错执行的序列一致，从而高效解决多变体执行的误报问题，具体是在编译阶段对产生误报的不确定性因素和关键代码地址进行分析，插桩到前驱函数与后驱函数，在运行时监视信号的传递，再结合主从变体同步机制，由主执行体将执行结果或顺序复制给从执行体。

通过动态链接[55]的方式在 sync ops 的指令前后插桩[56]before_sync_op 、after_sync_op 函数，图 9 所示，黑色代码为源代码，红色部分为插桩后的。插桩的同步代理为动态链接库，由 LD_PRELOAD 环境变量指示在运行时加载链接到程序中。在加载过程中，不同执行体的同步代理通过进程间的通信接口挂载到同步缓冲区，即 sync buffer。主执行体的同步代理在 sync buffer 中记录执行 sync op 的顺序，从执行体查询 sync op 的序列并控制 sync ops 的执行序列。

在插桩模块上，通过多样化编译技术来对软件程序内部进行精细化处理。

```
1.void spinlock_lock(int * ptr){
    bool result = false;
        while (!result){
        before_sync_op(ptr);
        result =compare_and_swap(ptr,0,1);
        after_sync_op(ptc);
        if (result) break;
        sched_yield();
    }
}
void spinlock_unlock(int *ptr){
    before_sync_op(ptr);
    *ptr=0;
    after_sync_op(ptr)
}
```

图 9　代码插桩示意图

Fig. 9　Schematic diagram of code instrumentation

3.2.1 全序同步方法

在用户空间内，申请一个缓冲区，该缓冲区串行运行互斥锁，当锁占用缓冲区资源时，需要等锁结束使用，当锁用[57]信号量 A，B 来标识同步的两个线程占用资源的状态，如 t1 时刻至 t2 时刻，线程 m1 执行系统调用和线程 s1 用信号量 A 来标识，一次执行过程开始 A 为 0，结束 A 为 1，m1 与 s1 对 A 进行互斥，一次执行结束后，线程 s1 开始读取缓冲区 m1 的调用顺序。同理 m2 与 s2 对 B 进行互斥，在同步阶段每个执行体同时只能执行一个系统调用，及达到多线程冗余执行的目的。全序同步图如图 10 所示。

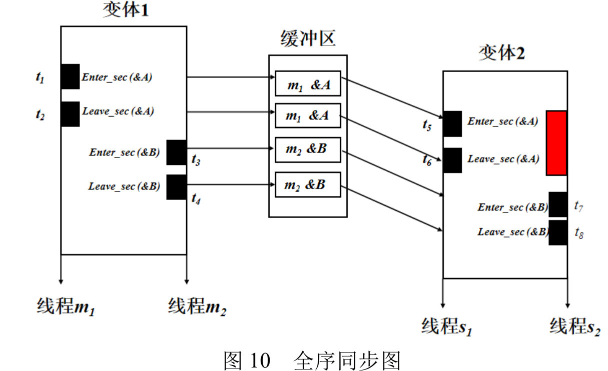


图 10　全序同步图

Fig. 10　Total sequence synchronization diagram

3.3　缩小表决边界的策略

多变体执行技术所带来的误报问题对该技术的可用性造成较为严重的影响，阻碍多变体发展，以往的多变体执行将普通系统改造成多变体系统后，先对该系统进行功能测试，接着查看表决器日志，分析测试中产生的报错，挑选出误报，对系统进行二次开发，填补误报“缺口”。但是多变体系统中包含多个异构冗余的执行体，填补一个“缺口”，需要改造多个执行体，耗费大量资源，并不能长时间有效的解决误报问题。

潘传幸等人提出冗余异构边界[58]的概念，通常过大的冗余异构边界会使得改造者对一些非必要的组件进行异构冗余处理，这会造成表决误报。邵昱文[59] 等人提出选择完全

冗余异构组件和最佳冗余异构集合的方法，以异构冗余为基础，缩小表决边界，对一些非必要的冗余异构组件进行删减，直至删减为误报最少的系统，则该系统所表示的集合为最佳冗余异构组件集。有些组件可以不进行异构冗余，而是兼容性改造，这样就可以避免一些冗余异构改造时带来的误报问题。

邵昱文等人基于王立群团队提出的面向非相似余度信息系统[60]的攻击面模型，利用攻击效费比[61]来度量系统的安全性，将多变体系统的输入与输出集合、通道传输集合、不可信数据项组成的三元组定义成攻击面，形式化表示

$$\mathrm{surf}\langle M,C,I\rangle \tag{1}$$

对于该攻击面 surf，其度量结果可以表示为

$$\left\langle \sum_{m\in M}\mathrm{der}(m),\ \sum_{c\in C}\mathrm{der}(c),\ \sum_{d\in I}\mathrm{der}(d) \right\rangle \tag{2}$$

其中，$\mathrm{der}(m)$，$\mathrm{der}(c)$ ，$\mathrm{der}(d)$ 分别表示系统入口点和出口点组件 、系统通道组件 、不可信数据项组件的攻击效费比。

系统的脆弱性可以表示为

$$vul=\sum_{m\in M}\mathrm{der}(m)+\sum_{c\in C}\mathrm{der}(c)+\sum_{d\in I}\mathrm{der}(d) \tag{3}$$

系统的安全性与系统的脆弱性[62]成负相关，系统的脆弱性越大，则系统的安全性越小。

systemA 表示没有继续进行多变体改造的系统，systemB 表示多变体改造后的系统。E 表示 systemA 中的所有组件的集合。E0 表示最佳冗余异构组件集，EH 表示完全冗余异构组件集。surfA 表示多变体执行改造前系统的攻击面，surfB 表示拟态化改造后系统的攻击面。

a)先求出改造后系统的安全增益[58]。

$$sec_{max}=vul_A-vul_B=\left(\sum_{m\in M_A}\mathrm{der}(m)+\sum_{c\in C_A}\mathrm{der}(c)+\sum_{d\in I_A}\mathrm{der}(d)-\left(\sum_{m\in M_B}\mathrm{der}(m)+\sum_{c\in C_B}\mathrm{der}(c)+\sum_{d\in I_B}\mathrm{der}(d)\right)\right) \tag{4}$$

b) 将 E0 赋值为 EH

$$E_0=E_H \tag{5}$$

c)从 E0 中任意取出一个组件 p，surf0 表示取出组件以后系统的攻击面。

此时的安全增益为

$$sec_o=vul_A-vul_o=\left(\sum_{m\in M_A}\mathrm{der}(m)+\sum_{c\in C_A}\mathrm{der}(c)+\sum_{d\in I_A}\mathrm{der}(d)\right)-\left(\sum_{m\in M_o}\mathrm{der}(m)+\sum_{c\in C_o}\mathrm{der}(c)+\sum_{d\in I_o}\mathrm{der}(d)\right) \tag{6}$$

判断 sec0 与 secmax 大小，若 sec0 小于 secmax，则将 p 组件放回到 E0，反之，则舍去 p 组件。

d)不断重复步骤 c)，直至将 E0 内的所有组件都取一遍，此时 E0 中的组件即为最佳冗余异构组件集。

表 2　代理服务器与执行体的具体配置

Tab. 2　Specific configuration of proxy server and executive body

	代理服务器	执行体 1	执行体 2	执行体 3
系统类型	Windows10	Ubuntu16	Centos7.0	Windows7
服务器类型	Nginx	Apache	Nginx	Iss
PHP 版本		PHP5.6	PHP5.5	PHP7.0
PHP 标签		PHP 标签 1	PHP 标签 2	PHP 标签 3
数据库标签		数据库标签 1	数据库标签 2	数据库标签 3

该方法以改造商用网站为例，选用 php 脚本组件，数据库服务组件，操作系统组件，网页服务器组件为组件集 E0，不断执行步骤四，选出最佳冗余异构组件集。首先选择需要改造的组件，然后再对商用网站系统进行冗余异构改造。通过访问日志分析，将用户正常请求(5000 次任意请求)中的误报率作为参考依据。

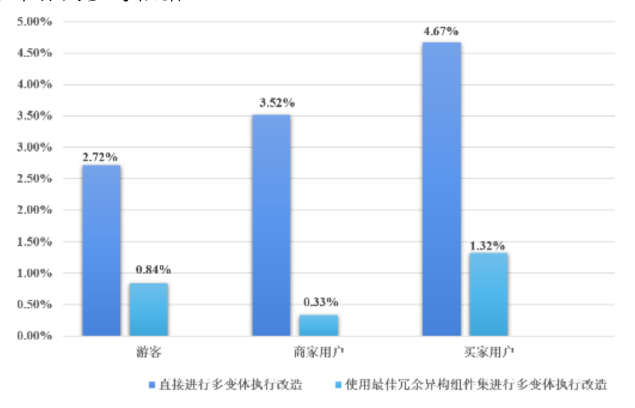


图 11　商用网站误报率统计

Fig. 11　Statistics of false alarm of commercial websites

由图 11 可看出，在不影响安全性的前提下合理缩小拟态界，可以有效降低甚至消除误报率。

3.4　对比实验

本实验选取 CentOS 7.3 系统为环境，建立三套同样硬件配置的虚拟机进行测试，其中环境一采取 DSL 策略，环境二采取编译器模块插桩的策略，环境三采取缩小表决边界的策略，三种环境均采用三路执行体和一路表决器。性能测试环境如表 3 所示。

表 3　环境硬件配置

Tab. 3　Environment hardware configuration

条目	参数
CPU	Intel(R) Xeon® CPU E5-2603 v4@ 1.70 GHz 6 cores
内存	32GB
操作系统	CentOS 7.3

对第二章提到的四种产生误报的应用场景分别测试，分析执行过程中表决器的表决日志，在每种对应的应用场景下，人工分析报错条目，挑选出符合类型的误报条目。如选择多线程程序执行的场景进行误报对比，在每个环境的表决器裁决日志里选取 1000 个报错条目，排除死锁等真实错误的条目，在剩下的条目里面，若没有影响系统功能正常执行，则认为这些条目是误报。误报统计如表 4 所示。

表 4　误报统计

Tab. 4　False alarm statistics

环境名	利用方法	多线程乱序执行	操作系统文件读写	验证随机数
环境一	DSL 策略	211(条)	5(条)	23(条)
环境二	编译器插桩策略	12(条)	3(条)	25(条)
环境三	缩小拟态界策略	45(条)	3(条)	2(条)

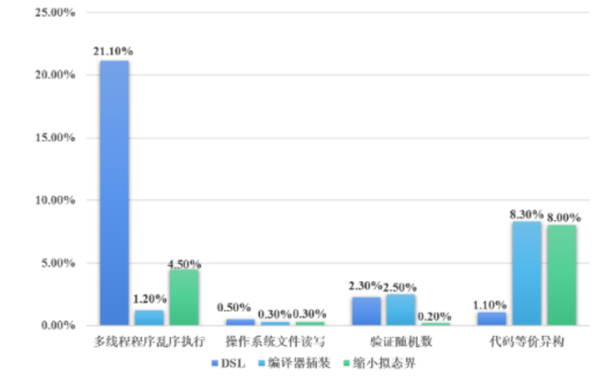


图 12　三种策略误报率对比

Fig. 12　Comparison chart of false positive rate of three strategies

由图 12 可以看到，在多线程乱序执行的场景下，编译器插桩的方法有效的减少多线程执行过程中产生的误报，能够同步执行体之间的线程调用顺序，DSL 的方法显然没有考虑到多线程执行情况，产生误报，线程一旦乱序执行，就不能语法匹配，同样缩小表决边界的方法也可以规避部分多线程程序乱序误报。

在操作系统文件读写的场景下，让三个测试环境分别 1000 次打开文件，写入文件，删除文件，虽然能产生进程号 pid 不一致等情况，但多路执行体通过接口端口转发给表决器，进程号本身就不会产生太多误报，因此三种方法都没有很好的解决此类误报，体现不出来各自的优势。

在验证随机数的应用场景下，Openssl 模块产生的随机数也是高频发生误报的场景，缩小拟态界的方法找出最佳冗余异构集合明显可以缩小产生随机数种子的边界，有效减少随机数的误报，相比之下，DSL 方法和编译器插桩方法效果一般。

在代码等价异构的应用场景下，DSL 可以有效的判断出代码等效，减少来自该场景的误报，而编译器插桩与缩小表决边界没有考虑到代码异构产生的误报。

4 误报解决策略的分析

在多变体执行架构中，表决器裁决误报问题一直是工程实现中的重大问题，三种解决误报问题方案，从不同粒度，不同应用场景各自解决部分误报问题，但自身还存在一定的缺陷，比如 DSL 策略是 2014 年提出的，当时没有考虑到多线程程序的执行可能产生误报，不适用于应用于多线程程序的场景。

4.1 功能性分析

本节对能否有效解决产生的误报问题作为功能性的主要评价，评测三种解决方案是否达到功能性目的，主要依据是否实现多变体执行系统完整功能，同时减少误报率，且该方法是否能推广应用于改造创新其他多变体架构。

首先，DSL 策略在软件层面实现多变体执行同步问题，设计算法对操作指令进行语义判断，提出一种从成体系的系统调用中自动提取相应规则的算法，提供冗余执行所需的操作，且实验证明该方法可以解决不同版本间的执行体，代码等价异构的问题，当两个版本执行体出现分歧，可能发出不同的系统调用或者某些不同的非确定性的参数，监视器会发出警告并停止执行或终止分歧版本，该方法可以用于同一程序的不同版本；并且与用于动态分析的版本并行运行原生版本，此方法能在语义判断上，对执行体歧义语句做处理，有效的避免误报。

编译器模块插桩的方法中，传统的系统调用监控器[63,64]依赖特定的系统调用序列进行检查,且没有对系统调用的参数进行比对,容易遭受伪装攻击[65]，模仿攻击可能会执行一系列数十个系统调用，以逃避检测，找到这样的序列是很困难的。通常采用静态分析技术，禁用的安全关键型系统调用[66]，但方法成本巨大，会禁用一些功能函数。而编译器插桩同步方法是在应用程序中内嵌监视器和在独立进程中设置额外监视器，在系统调用级别执行监控和输入/输出复制操作。监控系统调用接口，默认实现所有的输入/输出操作都可以被监控和复制，所有潜在的危险操作都可以在该接口上停止，即可以提供必要的安全保证。可加载的内核模块实现的监控器拦截系统调用是最有效的,因为它不需要额外的上下文切换，本质是对线程共享资源进行加锁，阻塞线程的执行。

缩小表决边界的方法，通过建立攻击面集合，具体方法是用排列组合的方法构造合适的冗余异构组件集，通过控制一个组件变化，其余组件不变，用攻击效费比的公式来度量系统的安全及解决误报的能力。多变体执行的根本目标是提供可靠可信可用的功能。该方法的优势是能够解决各执行体中产生随机数误报的场景效果明显，能够有效规避随机数给系统带来的影响，且该策略适用面广，适用于大部分应用场景。

4.2 性能损耗分析

阻碍多变体执行系统应用于实践的另一个障碍是性能方面的考虑，首先，从性能的平衡上考虑，影响系统性能的关键是取决于执行体同步和拦截的粒度以及执行体与监控器之间的通信成本，监控对象的数量越多，监控同步点粒度越细，性能越高，但工作量也会随着变大，有的团队为更好的提升监控能力和安全性能，采用分布式多变体执行架构，如 Dmon 分布式多变体架构，Dmon 利用跨平台天然存在的多样性实现指令集机构执行，并利用 ARM 和 X86 不同的硬件安全机制，进一步提升多变体执行安全性，但运行 Dmon 的复制逻辑需要的昂贵的网络通信，还可能有被信道攻击的风险，除政府单位或少数机构，很少有企业公司会消耗两倍以至于两倍以上的成本去实现分布式多变体执行架构。所以，在考虑性能时，需要对多变体执行架构及系统安全进行度量，对系统自带的特征进行形象化描述，从中删除一些冗余属性，减小成本，其次一些团队在基于内核的多变体架构中，通过在执行体初始化时删除冗余的属性参数，或者把监视器嵌入到内核中等方式也可以很大程度提升整个架构性能。

在软件层面实现改造，DSL 策略本身不需要改变物理硬件，实现成本是三个方法里成本最小的，但设置程序同步断点难度很大，通过语言处理，形成一套语言处理规则，DSL 能够很容易地对两次执行发出的系统调用序列的差异进行比对，它将大幅度改善代码异构及单线程系统调用，但此方法也有局限性，该方法的提出是为解决误报问题，在语义判断上添加自己的一套算法，但在复杂的多变体执行架构的执行过程中如发生未添加的语义等价漏洞，则需要改变整个系统。

对于编译器插桩的同步方法，本质是对线程共享资源进行加锁，阻塞线程的执行必然会导致性能的降低，但该方法能有效解决执行体间系统调用顺序不同导致的误报问题，局限在于对于动态申请的内存空间无法跟踪，因为所有的变量和地址信息都是静态分析和插桩的，虽然解决执行体之间系统调用顺序问题，能有效的避免误报，但加入大量的锁机制必然会导致性能的降低。

在缩小表决边界的方法里，实现成本体现在选取最佳冗余异构组件集上，攻击效费比度量是一种筛选策略，并不消耗硬件成本，甚至可以对需要改造的组件进行精简，减少系统开发的损耗。在软件层面，该策略通过增补或删减拟态组件，不需要对各执行体的系统调用级粒度行为或者线程级粒度行为进行同步，不需要改变整个系统，比 DSL 和编译器插桩实现要容易。缺点在于在选取组件时没法进行很好的安全度量，计算安全增益很困难。

5 结束语

5.1 思路总结

关于多变体执行误报问题，自 06 年多变体架构问世以来，一直都是一个难题，执行体中的不确定性及不一致的参数或状态会影响表决器表决。这类问题一方面来自于多线程程序以及子进程和线程的调度问题,另一方面来自异步信号、文件描述符、进程 ID、时间和随机数不一致,这些都会引起表决器裁决时的误报。本论文针对这些情况，对一些实际应用中产生的误报问题进行举例，并总结了三种不同类型的解

决误报的方法，横向对比这三种方法，给出了各自的优点及缺点。三个方法对减少多变体执行误报的研究有极大的启发意义。本文详细举出了多变体执行产生误报问题的产生案例及原因,对比近年来国内外团队研究设计解决误报问题的策略优缺点,指出在工程实现时需要注意的功能性及性能考虑。

多变体执行引入多线程冗余执行的场景一直是研究的重点，对于解决多线程冗余执行通常有两种思路：1.确定性多线程的系统，该系统通过为每个给定的程序输入建立一个固定的时间表来对线程间的通信指令施加一个确定的顺序,确定性多线程模式，将程序分为并行和串行阶段,串行阶段：线程共享操作的执行阶段被归类为串行阶段，任何时刻只有一个线程在执行,并行阶段：无共享操作的阶段线程并行执行由硬件计数器来确定程序执行的原子起点和终点。2. 记录+重放模式的系统[67]，在运行时执行记录+重放的操作，分为一个主执行体，多个从执行体，从变体通过捕获主执行体中执行同步操作的顺序，并在其他执行体中强制执行相同的顺序。在近几年的研究中，围绕记录+重放的研究成为多变体执行技术的重点研究之一，因为这样监控拦截粒度更细，效率更高，能够提升系统整体效能。

5.2 现有技术未解决的难点

虽然三种策略可以有效消除执行体之间部分误报的问题，但仍有部分情况会导致误报。实际的系统调用通常不会在同一时间执行。例如：执行体请求以只读方式打开文件。如果这些文件中的任何一个文件在某个执行体读取它之后被其他执行体读取它之前被第三方应用程序更改，则存在竞争关系，并且执行体将接收到不同的数据，这将导致它们之间出现差异。如果执行体尝试直接用处理器时间戳计数，例如使用 X86 处理器可用的 RDTSC[68] 指令，则会触发误报。因为读取时间戳的计数器是在没有任何系统调用调用的情况下执行的，因此不会通知监视器并且无法替换执行体收到的结果。

5.3 未来的展望

未来的多变体执行产生的误报问题的研究应多考虑避免引入多余的安全问题，结合近年来新兴的攻击手段，如 Linux 等常用的操作系统漏洞、信道攻击漏洞[69,70]，Openssl 模块漏洞等，从安全性的角度去设计。在效率层面，可以从编译支持的多变体执行思路入手，通过提前对产生误报的非一致性参数及地址进行分析提取，设计同步库，插桩到前驱函数及后驱函数，再根据主从同步机制将主执行体的执行结果或调用顺序等关键操作复制给从执行体。还可以从容器的角度入手，容器本质上是资源隔离的进程，其通过内核机制 Namespace 和 Cgroup[71,72]进行进程资源的隔离和限制，等价的容器[72]进程各自拥有独立的文件系统视图，不会因为文件描述符指向的目标点文件不同而发生误报。且容器与宿主机共享操作系统内核，在使用的资源和速度方面远远优于传统的虚拟机，但容器也存在缺点，虽然可以解决文件描述符等系统共享资源的误报，但遇到使用随机数的情况以及多线程程序的线程随机调度的情况时，暂时无法解决，还需要进一步研究。

致 谢 本文工作受到国家自然科学基金项目(61521003)的资助，以及感谢网络通信与安全紫金山实验室的资助。

参考文献：

[1] 姚东，张铮，张高斐，等. 多变体执行安全防御技术研究综述 [J]. 信息安全学报, 2020, 5 (5): 77-94. (Yao Dong, Zhang Zheng, Zhang Gaofei, *et al.* Summary of Research on Multi-variant Execution Security Defense Technology [J]. Journal of Cyber Security, 2020, 5 (5): 77-94.)

[2] Salamat B. “Multivariant program execution: Using multi-core systems to defuse buffer-overflow vulnerabilities, ”Proc. -CISIS 2008 2nd Int. Conf. Complex, Intell. Softw. Intensive Syst. pp. 843–848, 2008, doi: 10. 1109/CISIS. 2008. 136.

[3] Salamat B. “Reverse stack execution in a multi-variant exe-cution environment”Work. Compil. Archit. Tech. Appl. Reliab. Secure, pp. 1-7, 2008.

[4] 陈平. 代码复用攻击与防御技术研究 [D]. 南京：南京大学, 2012. (Chen Ping. Research on the Attack and Defense Techniques of Code Reuse [D] , Nanjing University, 2012.)

[5] 柳童，史岗，孟丹. 代码重用攻击与防御机制综述 [J]. 信息安全学报, 2016, 1 (2): 15-27. Liu Tong, Shi Gang, Meng Dan. A Survey of Code Reuse Attack and Defense Mechanisms [J]. Journal of Cyber Security, 2016, 1 (2): 15-27.

[6] Tran M, Etheridge M, Bletsch T, *et al.* On the expressiveness of return-into-libc attacks [C]// International Workshop on Recent Advances in Intrusion Detection. Springer, Berlin, Heidelberg, 2011: 121-141.

[7] 邬江兴. 网络空间拟态防御研究 [J]. 信息安全学报, 2016 (4): 1-10. (Wu Jiang Xin. Research on Cyber Mimic Defense [J]. Journal of Cyber Security, 2016, 1 (4): 1-10.)

[8] 姚东，张铮，张高斐，等. MVX-CFI: 一种实用的软件安全主动防御架构 [J]. 信息安全学报, 2020, 5 (4): 44-54. (Yao Dong, Zhang Zheng, Zhang Gaofei, *et al.* MVX-CFI: a practical active de-fense framework for software security [J]. Journal of Cyber Security, 2020, 5 (4): 44-54.)

[9] 邬江兴. 网络空间拟态防御原理：广义鲁棒控制与内生安全 (上 册) [M]. 北京：科学出版社, 2018. (Wu Jiang Xin. Principles of mimic defense in cyberspace: generalized robust control and endogenous security (Volume 1) [M]. Beijing: Science Press, 2018.)

[10] 邬江兴. 网络空间拟态防御原理：广义鲁棒控制与内生安全 (下 册) [M]. 北京：科学出版社, 2018. (Wu Jiang Xin. Principles of mimic defense in cyberspace: generalized ro bust control and endogenous security (Volume 2) [M]. Beijing: Science Press, 2018.)

[11] 马海龙，伊鹏，江逸茗. 基于动态异构冗余机制的路由器拟态防御体系结构 [J]. 信息安全学报, 2017, 2 (01): 29-42. Ma Hai Long, Yi Peng Jiang Yi Min. Router mimic defense architecture based on dynamic heterogeneous redundancy mechanism [J]. Journal of Cyber Security, 2017, 2 (01): 29-42.

[12] 吴铤. 基于执行体划分的防御增强型动态异构冗余架构 [J]. 通信学报 2021, 42 (3): 122-134. (Wu Ting. Defense enhanced dynamic heterogeneous redundant architecture based on executive body division [J]. Journal of Communications, 2021, 42 (3): 122-134.)

[13] 仝青，张铮，张为华，等. 拟态防御 Web 服务器设计与实现 [J]. 软件学报, 2017, 28 (4) . (Tong Qing, Zhang Zheng, Zhang Weihua, Wu Jiangxing. Design and implemention of mimic defense Web server [J]. Ruan Jian Xue Bao/Journal of Software, 2017, 28 (4): 883-897.)

[14] 张明武，沈华，穆怡. 虚拟黑盒安全的程序混淆：模型，进展与挑战 [J]. 计算机学报, 2017, 40 (12): 2700-2718. (Zhang MingWu, ShenHua, MuYi. Program confusion for virtual black box security: models, progress and challenges [J]. Journal of Computers, 2017, 40 (12): 2700-2718.)

[15] Kuppa A, Le-Khac N A. Black box attacks on explainable artificial intelligence (XAI) methods in cyber security [C]// 2020 International Joint Conference on Neural Networks (IJCNN) . IEEE, 2020: 1-8.

[16] Kalin J, Ciolino M, Noever D, *et al.* Black Box to White Box: Discover Model Characteristics Based on Strategic Probing [C]// 2020 Third International Conference on Artificial Intelligence for Industries (AI4I) . IEEE, 2020: 60-63.

[17] Shacham H. The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86) [C]// Proceedings of the 14th ACM conference on Computer and communications security. 2007: 552-

561.

[18] Barrantes E G, Ackley D H, Forrest S, *et al.* Randomized instruction set emulation to disrupt binary code injection attacks [C]// Proceedings of the 10th ACM conference on Computer and communications security. 2003: 281-289.

[19] Sinha K, Kemerlis V P, Sethumadhavan S. Reviving instruction set randomization [C]// 2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST) . IEEE, 2017: 21-28.

[20] Guanciale R. Protecting instruction set randomization from code reuse attacks [C]// Nordic Conference on Secure IT Systems. Springer, Cham, 2018: 421-436.

[21] Cox B, Evans D, Filipi A, *et al.* N-Variant Systems: A Secretless Framework for Security through Diversity [C]// USENIX Security Symposium. 2006: 105-120.

[22] Bala V, Duesterwald E, Banerjia S. Dynamo: A transparent dynamic optimization system [C]// Proceedings of the ACM SIGPLAN 2000 conference on Programming language design and implementation. 2000: 1-12.

[23] Salamat B. Multi-Variant Execution: Run-Time Defense against Malicious Code Injection Attacks DISSERTATION [D]. Ph. D. Dissertation. University of California, Irvine, 2009.

[24] Holland D A, Lim A T, Seltzer M I. An architecture a day keeps the hacker away [J]. ACM SIGARCH Computer Architecture News, 2005, 33 (1): 34-41.

[25] Bruschi D, Cavallaro L, Lanzi A. Diversified process replicæ for defeating memory error exploits [C]// 2007 IEEE International Performance, Computing, and Communications Conference. IEEE, 2007: 434-441.

[26] Berger E D, Zorn B G. DieHard: Probabilistic memory safety for unsafe languages [J]. Acm sigplan notices, 2006, 41 (6): 158-168.

[27] Saito T, Watanabe R, Kondo S, *et al.* A survey of prevention/mitigation against memory corruption attacks [C]// 2016 19th International Conference on Network-Based Information Systems (NBiS) . IEEE, 2016: 500-505.

[28] Conti M, Crane S, Davi L, *et al.* Losing control: On the effectiveness of control-flow integrity under stack attacks [C]// Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. 2015: 952-963.

[29] Cavallaro L. Comprehensive memory error protection via diversity and taint-tracking [D]. PhD thesis, PhD dissertation, Universita Degli Studi Di Milano, 2007.

[30] Salamat B, Jackson T, Gal A, *et al.* Orchestra: intrusion detection using parallel execution and monitoring of program variants in user-space [C]// Proceedings of the 4th ACM European conference on Computer systems. 2009: 33-46.

[31] Volckaert S, De Sutter B, De Baets T, *et al.* GHUMVEE: efficient, effective, and flexible replication [C]// International Symposium on Foundations and Practice of Security. Springer, Berlin, Heidelberg, 2012: 261-277.

[32] Maurer M, Brumley D. TACHYON: Tandem execution for efficient live patch testing [C]// 21st{USENIX}Security Symposium ({USENIX}Security 12) . 2012: 617-630.

[33] Hosek P, Cadar C. Safe software updates via multi-version execution [C]// 2013 35th International Conference on Software Engineering (ICSE) . IEEE, 2013: 612-621.

[34] Hosek P, Cadar C. Varan the unbelievable: An efficient n-version execution framework [J]. ACM SIGARCH Computer Architecture News, 2015, 43 (1): 339-353.

[35] Hu H, Shinde S, Adrian S, *et al.* Data-oriented programming: On the expressiveness of non-control data attacks [C]// 2016 IEEE Symposium on Security and Privacy (SP) . IEEE, 2016: 969-986.

[36] Volckaert S, Coppens B, Voulimeneas A, *et al.* Secure and efficient application monitoring and replication [C]// 2016{USENIX}Annual Technical Conference ({USENIX}{ATC}16) . 2016: 167-179.

[37] Volckaert S, Coppens B, De Sutter B. Cloning your gadgets: Complete ROP attack immunity with multi-variant execution [J]. IEEE Transactions on Dependable and Secure Computing, 2015, 13 (4): 437-450.

[38] Belay A, Bittau A, Mashtizadeh A, *et al.* Dune: Safe user-level access to privileged{CPU}features [C]// 10th{USENIX}Symposium on Operating Systems Design and Implementation ({OSDI}12) . 2012: 335-348.

[39] Koning K, Bos H, Giuffrida C. Secure and efficient multi-variant execution using hardware-assisted process virtualization [C]// 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN) . IEEE, 2016: 431-442.

[40] Lu K, Xu M, Song C, *et al.* Stopping memory disclosures via diversification and replicated execution [J]. IEEE Transactions on Dependable and Secure Computing, 2018.

[41] Österlund S, Koning K, Olivier P, *et al.* kMVX: Detecting kernel information leaks with multi-variant execution [C]// Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems. 2019: 559-572.

[42] Voulimeneas A, Song D, Parzefall F, *et al.* DMON: A Distributed Heterogeneous N-Variant System [J]. arXiv preprint arXiv: 1903. 03643, 2019.

[43] Wang X, Yeoh S M, Olivier P, *et al.* Secure and efficient in-process monitor (and library) protection with intel mpk [C]// Proceedings of the 13th European workshop on Systems Security. 2020: 7-12.

[44] 张宇嘉，庞建民，张铮，等. 基于软件多样化的拟态安全防御策略 [J]. 计算机科学, 2018, 45 (2): 215-221. (Zhang Yu Jia, Pang Jian Ming, Zhang zhen, Wu Jiang Xing. A mimic security de fence strategy based on software diversity [J]. Computer Science, 2018, 45 (2): 215-221.)

[45] 庞建民，张宇嘉，邬江兴，等. 拟态防御技术结合软件多样化在软件安全产业中的应用 [J]. 中国工程科学, 2016, 18 (6): 74-78. (Pang Jian Ming, Zhang Yu Jia, Wu Jiang Xing *et al.* Applying a combination of mimic defense and software diversity in the software security indus try [J]. Strategic Study of Chinese Academy of Engineering, 2016, 18 (6): 74-78.)

[46] 潘传幸，张铮，马博林，等. 面向进程控制流劫持攻击的拟态防御方法 [J]. 通信学 报, 2021, 42 (1): 37-47. (Pan Chuanxin, Zhengzhen, Mabolin *et al.* Method against process control-flow hijacking based on mimic defense [J]. Journal on Communications, 2021, 42 (1): 37-47.)

[47] Gras B, Razavi K, Bosman E, *et al.* ASLR on the Line: Practical Cache Attacks on the MMU [C]// NDSS. 2017, 17: 26.

[48] Viega J, Messier M, Chandra P. Network security with openSSL: cryptography for secure communications [M]. "O'Reilly Media, Inc. ", 2002.

[49] Jimenez M, Papadakis M, Le Traon Y. An empirical analysis of vulnerabilities in openssl and the linux kernel [C]// 2016 23rd Asia-Pacific Software Engineering Conference (APSEC) . IEEE, 2016: 105-112.

[50] 曹志波. OpenSSL 的心脏出血漏洞 [J]. 电子技术与软件工程, 2017 (13): 263-263. (Cao Zhi Bo. Heartbleed Vulnerability in OpenSSL [J]. Electronic Technology and Software Engineering. 2017 (13): 262-263.)

[51] 张宇嘉，张啸川，庞建民. 代码混淆技术研究综述 [J]. 信息工程大学学报, 2017, 18 (5): 635-640. (Zhang Yu Jia, Zhang Xiao Chuan, Pang

Jian Ming. Survey on code obfuscation research [J]. Journal of Information Engineering University, 2017, 18 (5): 635-640)

[52] Pina L, Grumberg D, Andronidis A, *et al.* A{DSL}Approach to Reconcile Equivalent Divergent Program Executions [C]// 2017{USENIX}Annual Technical Conference ({USENIX}{ATC}17) . 2017: 417-429.

[53] Harvan M, Pretschner A. State-based usage control enforcement with data flow tracking using system call interposition [C]// 2009 Third International Conference on Network and System Security. IEEE, 2009: 373-380.

[54] Volckaert S, Coppens B, De Sutter B, *et al.* Taming parallelism in a multi-variant execution environment [C]// Proceedings of the Twelfth European Conference on Computer Systems. 2017: 270-285.

[55] Ho W W, Olsson R A. An approach to genuine dynamic linking [J]. Software: Practice and Experience, 1991, 21 (4): 375-390.

[56] Roemer R, Buchanan E, Shacham H, *et al.* Return-oriented programming: Systems, languages, and applications [J]. ACM Transactions on Information and System Security (TISSEC) , 2012, 15 (1): 1-34.

[57] Lamport L. Time, clocks, and the ordering of events in a distributed system [M]// Concurrency: the Works of Leslie Lamport. 2019: 179-196.

[58] 姚远，潘传幸，张铮，等. 多样化软件系统量化评估方法 [J]. 通信学报, 2020, 41 (3): 120-125. Yao yuan, pan Chuanxing, Zhang Zheng, *et al.* Quantitative evaluation method of diversified software systems [J]. Acta Telecom Sinica, 2020, 41 (3): 120-125

[59] Yuwen S, Zheng Z, Bingzheng L, *et al.* A Multi-Variant Voting Algorithm Based on Dynamic Feedback [C]// 2021 2nd International Conference on Computer Communication and Network Security (CCNS) . IEEE, 2021: 134-140.

[60] 张铮，王立群，李卫超. 面向非相似余度信息系统的攻击面模型 [J]. 通信学报, 2018, 39 (S2): 227-234. (Zhang Zheng, Wang Li Qun, Li Wei Cheng. Research on formal model for an information system' s attack surface with dissimilar redundant architecture. Journal of Communications [J] , 2018, 39 (S2): 227-234.)

[61] 邬江兴. 内生安全：重新定义新基建的安全属性 [J]. 中国科技产业, 2020 (5): 7-9. (Wu Jiangxing. Endogenous security: redefining the security attribute of new infrastructure [J]. China Science and technology industry, 2020 (5): 7-9)

[62] 李卫超，张铮，王立群. 基于拟态防御架构的多余度裁决建模与风险分析 [J]. 信息安全学报, 2018, 3 (5): 64-74. (Li Wei Chao, Zheng Zheng, Wang Li Qun. Redundancy adjudication modeling and risk analysis based on mimic defense architecture [J]. Journal of Cyber Security, 2018, 3 (5): 64-74.)

[63] Sato M, Taniguchi H, Yamauchi T. Design and implementation of hiding method for file manipulation of essential services by system call proxy using virtual machine monitor [J]. International Journal of Space-Based and Situated Computing, 2019, 9 (1): 1-10.

[64] Garfinkel T, Pfaff B, Rosenblum M. Ostia: A Delegating Architecture for Secure System Call Interposition [C]// NDSS. 2004.

[65] Parampalli C, Sekar R, Johnson R. A practical mimicry attack against powerful system-call monitors [C]// Proceedings of the 2008 ACM symposium on Information, computer and communications security. 2008: 156-167.

[66] Ghavamnia S, Palit T, Mishra S, *et al.* Temporal system call specialization for attack surface reduction [C]// 29th{USENIX}Security Symposium ({USENIX}Security 20) . 2020: 1749-1766.

[67] Ronsse M, De Bosschere K. RecPlay: A fully integrated practical record/replay system [J]. ACM Transactions on Computer Systems (TOCS) , 1999, 17 (2): 133-152.

[68] Song C, Yongqing W. Detection and research of RTX timer actual computing workload for ONC system based on RDTSC [J]. The International Journal of Advanced Manufacturing Technology, 2011, 57 (1-4): 257.

[69] Evtyushkin D, Riley R, Abu-Ghazaleh N C S E E C E, *et al.* Branchscope: A new side-channel attack on directional branch predictor [J]. ACM SIGPLAN Notices, 2018, 53 (2): 693-707.

[70] Yarom Y, Benger N. Recovering OpenSSL ECDSA Nonces Using the FLUSH+RELOAD Cache Side-channel Attack [J]. IACR Cryptol. ePrint Arch. , 2014, 2014: 140.

[71] Rosen R. Resource management: Linux kernel namespaces and cgroups [J]. Haifux, May, 2013, 186: 70.

[72] Xing F, Zhang Z, Ma B, *et al.* Design and implementation of endogenous security container based on union file system [C]// Journal of Physics: Conference Series. IOP Publishing, 2021, 2078 (1): 012080.